© Fabrice Rossi, 1997-2001

Conditions de distribution et de copie

Cet ouvrage peut être distribué et copié uniquement selon les conditions qui suivent :

1. toute distribution commerciale de l'ouvrage est interdite sans l'accord préalable explicite de l'auteur. Par distribution commerciale, on entend une distribution de l'ouvrage sous quelque forme que ce soit en échange d'une contribution financière directe ou indirecte. Il est par exemple interdit de distribuer cet ouvrage dans le cadre d'une formation payante sans autorisation préalable de l'auteur ;
2. la redistribution gratuite de copies exactes de l'ouvrage sous quelque forme que ce soit est autorisée selon les conditions qui suivent :
   (a) toute copie de l'ouvrage doit impérativement indiquer clairement le nom de l'auteur de l'ouvrage ;
   (b) toute copie de l'ouvrage doit impérativement comporter les conditions de distribution et de copie ;
   (c) toute copie de l'ouvrage doit pouvoir être distribuée et copiée selon les conditions de distribution et de copie ;
3. la redistribution de versions modifiées de l'ouvrage (sous quelque forme que ce soit) est interdite sans l'accord préalable explicite de l'auteur. La redistribution d'une partie de l'ouvrage est possible du moment que les conditions du point 2 sont vérifiées ;
4. l'acceptation des conditions de distribution et de copie n'est pas obligatoire. En cas de non acceptation de ces conditions, les règles du droit d'auteur s'appliquent pleinement à l'ouvrage. Toute reproduction ou représentation intégrale ou partielle doit être faite avec l'autorisation de l'auteur. Seules sont autorisées, d'une part, les reproductions strictement réservées à l'usage privé et non destinées à une utilisation collective, et d'autre part, les courtes citations justifiées par le caractère scientifique ou d'information de l'oeuvre dans laquelle elles sont incorporées (loi du 11 mars 1957 et Code pénal art. 425).

Tables de hachage

1 Introduction

Le but de ce projet est la programmation d'une structure de données appelée table de hachage. Cette structure permet un stockage efficace d'information sous une forme proche d'un dictionnaire. Dans un tableau Java, on stocke les informations en les repérant par un numéro, leur indice dans le tableau. Au contraire, une table de hachage permet un repérage plus intuitif, théoriquement par n'importe quelle clé. Dans la pratique, on s'intéresse surtout au stockage d'informations repérées par une clé alphanumérique, le prototype étant le stockage d'une définition associée à un mot : le dictionnaire.

2 Un dictionnaire simple

2.1 Les données

En vous inspirant de votre cours, vous commencerez par définir une classe `Enregistrement` donc chaque instance permet de stocker une définition et une clé, représentées par des `String`s. Ainsi doit-on pouvoir créer un enregistrement grâce à la ligne suivante :

```
Enregistrement e=new Enregistrement("Voiture","Engin de mort, cher et polluant") ;
```

2.2 Le dictionnaire

Pour stocker un dictionnaire, il suffit en fait de stocker des `Enregistrement`s, ce qui peut se faire sans problème grâce à un tableau. Le dictionnaire sera donc un `Enregistrement[]`[1].

Vous écrirez alors des méthodes (de classe) regroupées au sein de la classe `Dictionnaire`, et permettant la manipulation du dictionnaire. On proposera en autre des méthodes permettant d'ajouter et de supprimer un `Enregistrement` à un dictionnaire (en produisant un nouveau tableau, c'est-à-dire un nouveau dictionnaire), de chercher une définition en ne connaissant que la clé, etc.

3 Principe du hachage

3.1 Motivation

Le principal défaut de la structure précédente est sa lenteur. Pour chercher une définition à partir de sa clé, il faut étudier tout le tableau. Le but des techniques de hachage est de remédier simplement à ce problème.

3.2 Fonction de hachage

Dans la suite de ce texte, nous allons utiliser une **fonction de hachage**. Dans le contexte des dictionnaires, une fonction de hachage est une fonction qui à une clé (c'est-à-dire une `String`) associe un entier positif ou nul. On notera `f` une telle fonction.

On suppose qu'à chaque fonction de hachage `f` est associée une "taille" t_f, un entier tel que pour toute clé `s`, $0 \leq f(s) < t_f$.

[1] Ce n'est bien sûr pas la meilleure façon de procéder. Il vaudrait mieux créer une classe `Dictionnaire`, mais ce n'est pas le but de ce projet.

3.3 Principe élémentaire des tables de hachage

On suppose donc donnée une fonction de hachage `f`, de taille t_f. Soit maintenant un enregistrement `e` constitué d'une clé `c` et d'une définition `d`. Normalement, si on dispose d'un dictionnaire `T`, on va écrire quelque chose de la forme `T[i]=e`, pour stocker l'enregistrement dans la première case libre de `T`.

Grâce à la fonction de hachage, on peut faire mieux. On peut en effet écrire `T[f(c)]=e`. Le principal intérêt de cette technique est le suivant : si on veut maintenant obtenir `d`, la définition associée à la clé `c`, il suffit de calculer `f(c)` pour savoir dans quelle case du dictionnaire l'enregistrement est rangé. Dans l'approche de la section précédente, il fallait regarder tous les enregistrements un par un afin de trouver le bon.

3.4 Difficulté

Le principal problème lié à cette approche est que deux clés différentes peuvent avoir la même image par la fonction de hachage, ce qui veut dire qu'on ne peut pas stocker directement un enregistrement dans la case `T[f(c)]`, car sinon on ne pourra plus stocker aucun autre enregistrement dont la clé est d'image égale à `f(c)`.

Pour remédier à ce problème, on stocke dans le tableau `T` des dictionnaires simples. Ceci signifie que le type de `T` est `Enregistrement[][]`. Si on appelle `n` la taille de `f`, on peut même dire que `T` va être créé par :

```
Enregistrement[][] T=new Enregistrement[][n] ;
```

Quand on souhaite ajouter un enregistrement `e` de clé `c`, on calcule donc `f(c)` et on ajoute `e` au dictionnaire `T[f(c)]`. De même, pour recherche une définition de clé `u`, il suffit de rechercher cette définition dans le dictionnaire `T[f(u)]`, ce qui doit aller plus vite que dans un dictionnaire complet.

4 Programmation

4.1 Fonction de hachage

Pour une taille `n` donnée, on obtient une fonction de hachage relativement efficace en associant à une chaîne de caractères le reste de la division par `n` de la somme des codes Unicode de ses caractères.

4.2 La table

Vous programmerez une nouvelle classe `Table` fonctionnant de façon similaire à `Dictionnaire`, mais utilisant la technique décrite dans la section précédente. Vous pourrez bien sûr utiliser la classe `Dictionnaire` que vous aurez déjà programmée.

5 Consignes générales

En cas de non respect d'une des règles suivantes, la note de projet sera divisée par deux :

1. le projet sera rendu sous la forme d'une disquette contenant les fichiers `.java` de votre programme, accompagnée d'une impression de ses fichiers et d'un mode d'emploi ;
2. **toutes** les méthodes devront être documentées selon la méthode décrite au chapitre 11 du polycopié de cours ;
3. le programme devra comporter une classe de démonstration des autres classes, permettant la manipulation de dictionnaires **déjà initialisés** et comportant au moins une trentaine d'enregistrement. Il devra être possible de modifier les dictionnaires (ajout et suppression) et de les interroger (recherche d'une définition) sans recompiler le programme.

Le projet devra être rendu le 6 avril avant 17h. Chaque jour de retard sera pénalisé par le retrait de 4 points à la note du projet. Aucune excuse ne sera acceptée, sauf grave problème médical dûment attesté par un certificat.